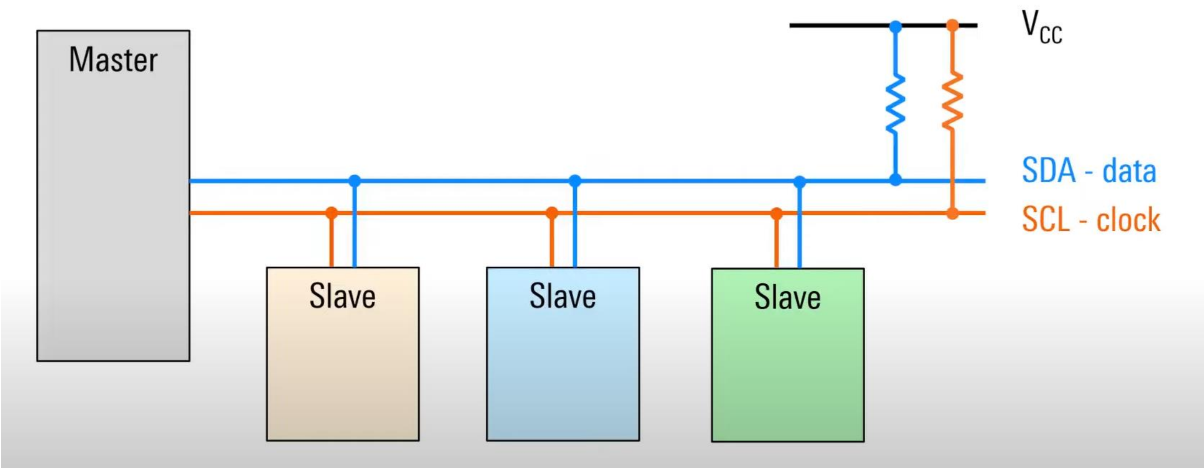
I2C Protocol with three different slaves



Created By:

Mohamed Ahmed Mohamed Hussein 11/12/2024

I2C Protocol Overview

Introduction

I2C (Inter-Integrated Circuit) is a synchronous, multi-master, multi-slave serial communication protocol developed by Philips Semiconductor (now NXP Semiconductors). It is widely used for short-distance communication between microcontrollers and peripherals in embedded systems. I2C is particularly popular due to its simplicity, scalability, and ability to connect multiple devices using only two wires.

Uses and Importance

1. **Efficient Peripheral Communication:**
   - Allows microcontrollers to communicate with peripherals like sensors, displays, EEPROMs, and more.
   - Scales easily to accommodate multiple devices on the same bus.
2. **Multi-Master and Multi-Slave Capability:**
   - Supports systems with multiple masters, though typically, a single master controls the communication.
   - Multiple slaves can coexist on the same bus, enabling system flexibility.
3. **Cost-Effective:**
   - Reduces the need for extensive wiring by using just two wires (SCL and SDA).
4. **Applications:**
   - Embedded systems (microcontrollers and sensors).
   - Consumer electronics (TVs, cameras, and phones).
   - Automotive (sensors and actuators).
   - Industrial control and IoT.

I2C Bus Signals

1. Serial Clock Line (SCL):

- **Purpose:** Carries the clock signal to synchronize communication between master and slaves.
- **Controlled By:** Typically driven by the master.
- **Behaviour:**
  - Alternates between high and low to create a clock signal.
  - Slaves cannot change their SDA signal while SCL is high (ensures data integrity).

2. Serial Data Line (SDA):

- **Purpose:** Transfers data between master and slaves.
- **Controlled By:** Both master and slave devices can pull SDA low to send data.
- **Behavior:**
  - Data changes on the falling edge of SCL.
  - Read or write operations depend on the state of SDA during data transfer.
- **Open-Drain Design:**
  - SDA is open-drain, meaning no device actively drives it high.
  - Pull-up resistors ensure the line is high when no device pulls it low.

Key Protocol Details

1. Addressing:

- Each slave device has a unique 7-bit or 10-bit address.
- The master sends the address to identify the target slave.
- Slaves remain silent unless addressed.

2. Data Transfer:

- Data is transmitted serially (bit-by-bit) over SDA, synchronized with SCL.
- Data direction (read/write) is determined by the R/W bit sent with the slave address.

3. Start and Stop Conditions:

- **Start Condition:** The master pulls SDA low while SCL is high.
- **Stop Condition:** The master releases SDA to high while SCL is high.

4. Acknowledge (ACK) and Not Acknowledge (NACK):

- After receiving each byte, the receiver (slave or master) acknowledges by pulling SDA low (ACK).
- If a device cannot process further, it sends NACK (SDA remains high).

5. Multi-Slave Communication:

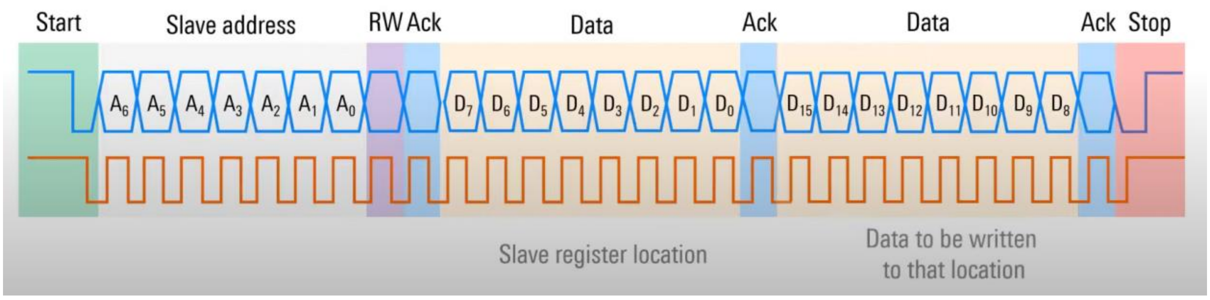
- Non-addressed slaves remain in a **high-impedance state**, ensuring no conflict on SDA or SCL.
- Only the addressed slave responds to the master.

6. Clock Stretching:

- A slave can hold SCL low to signal the master to wait, ensuring it has enough time to process data.

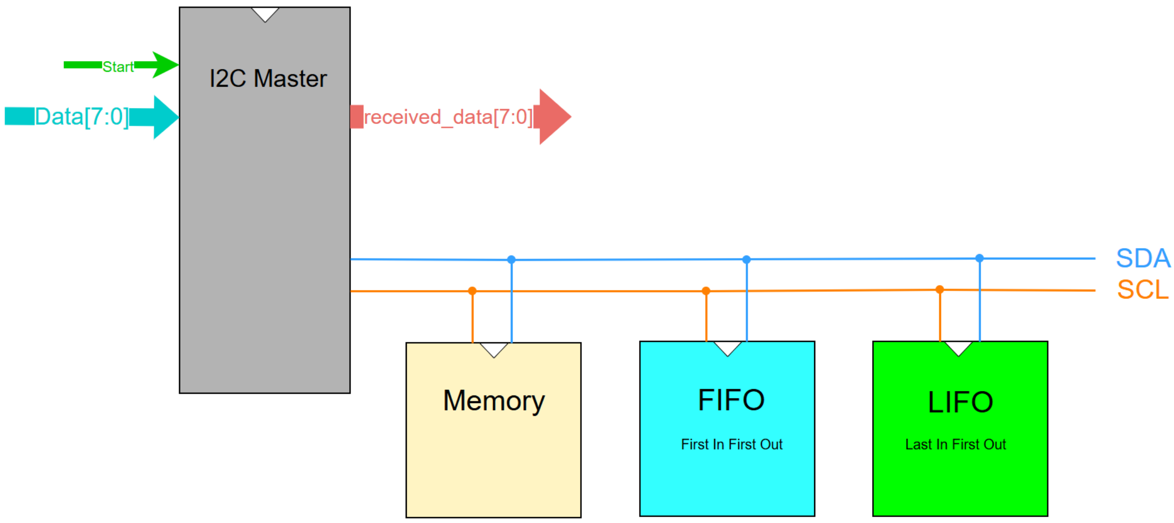
Our Implementation:

- ## Explanation of the flow:



1. **Start:**
   - The communication begins with the Start Condition.
2. **Slave Address:**
   - The master sends the slave's address (A6 to A0) and R/W bit.
3. **ACK from Slave:**
   - The slave acknowledges by pulling SDA low.
4. **Data Phase 1:**
   - The master sends the register location (D7 to D0).
   - The slave acknowledges.
5. **Data Phase 2:**
   - The master sends data to be written to that location (D15 to D8).
   - The slave acknowledges.
6. **Stop:**
   - The communication ends with the Stop Condition.

- ## Design:



Slaves Implemented:

- **Memory:** Stores data at specific addresses.
- **FIFO (First In, First Out):** Allows sequential data handling, useful for buffering.
- **LIFO (Last In, First Out):** Allows stack-like data handling.

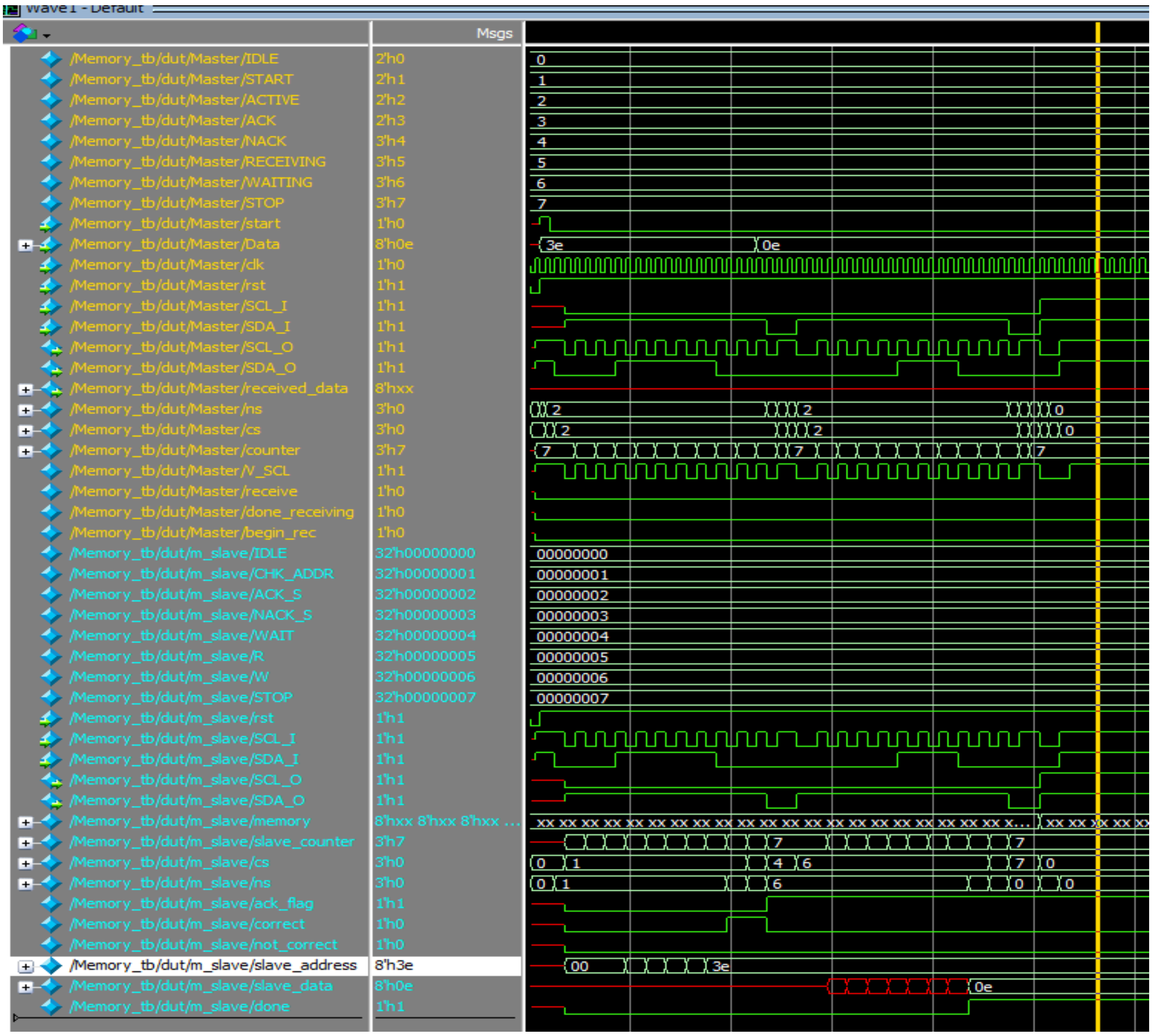
Notes:

1. **Verification Process for Each Slave:**
   - Each slave device (Memory, FIFO, and LIFO) will undergo **individual verification** to ensure its functionality and compliance with the I2C protocol.
2. **Implementation Using Vivado:**
   - After verification, each slave will be synthesized, implemented, and elaborated using **Vivado** to ensure successful hardware realization.
3. **Snapshots in QuestaSim:**
   - In **QuestaSim simulation snapshots**, the **Master** will be highlighted in **gold**, while the **Slave** (any of the three) will be highlighted in **blue** for clarity and differentiation.

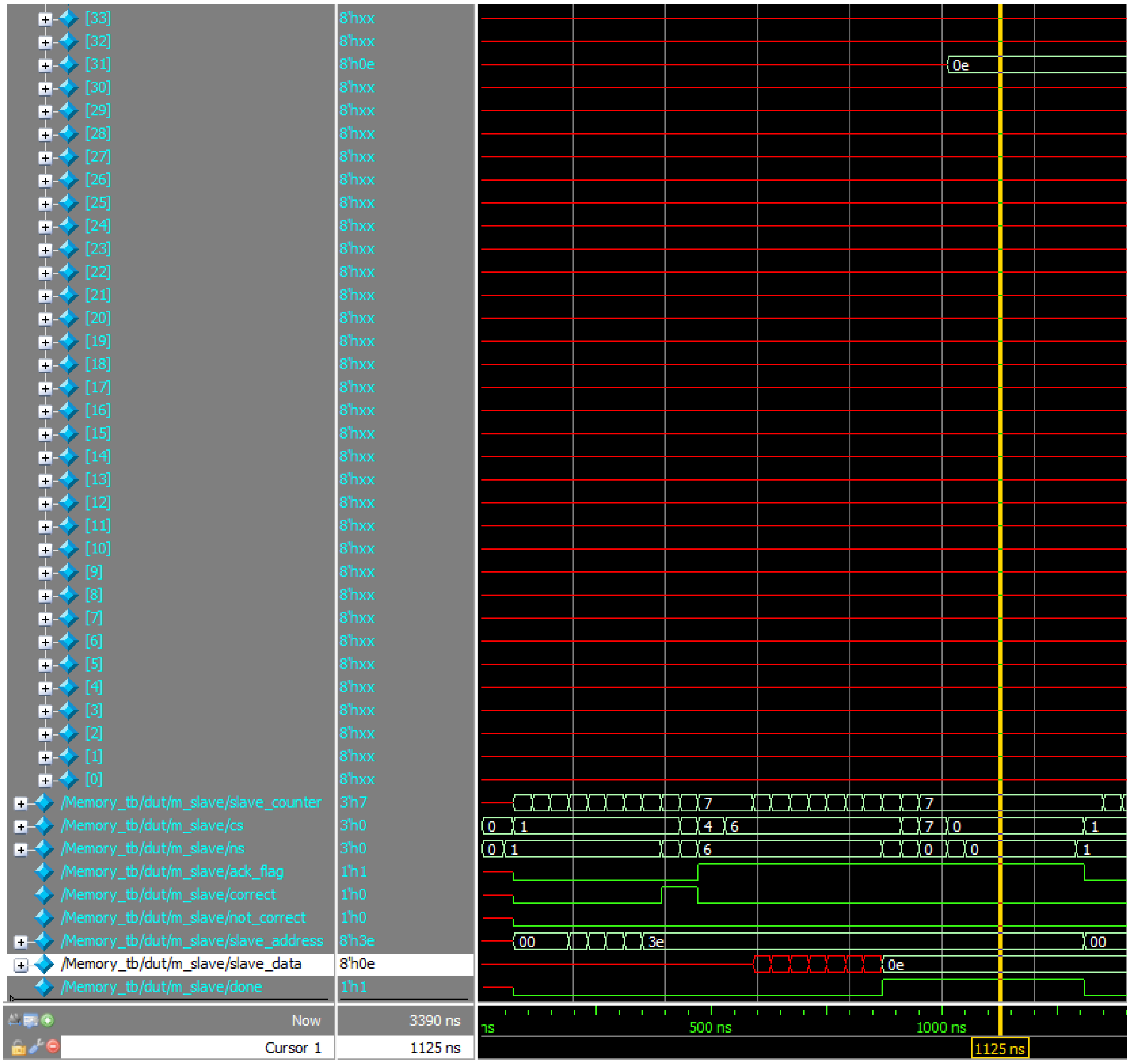
Memory

- ## Verifying Functionality:

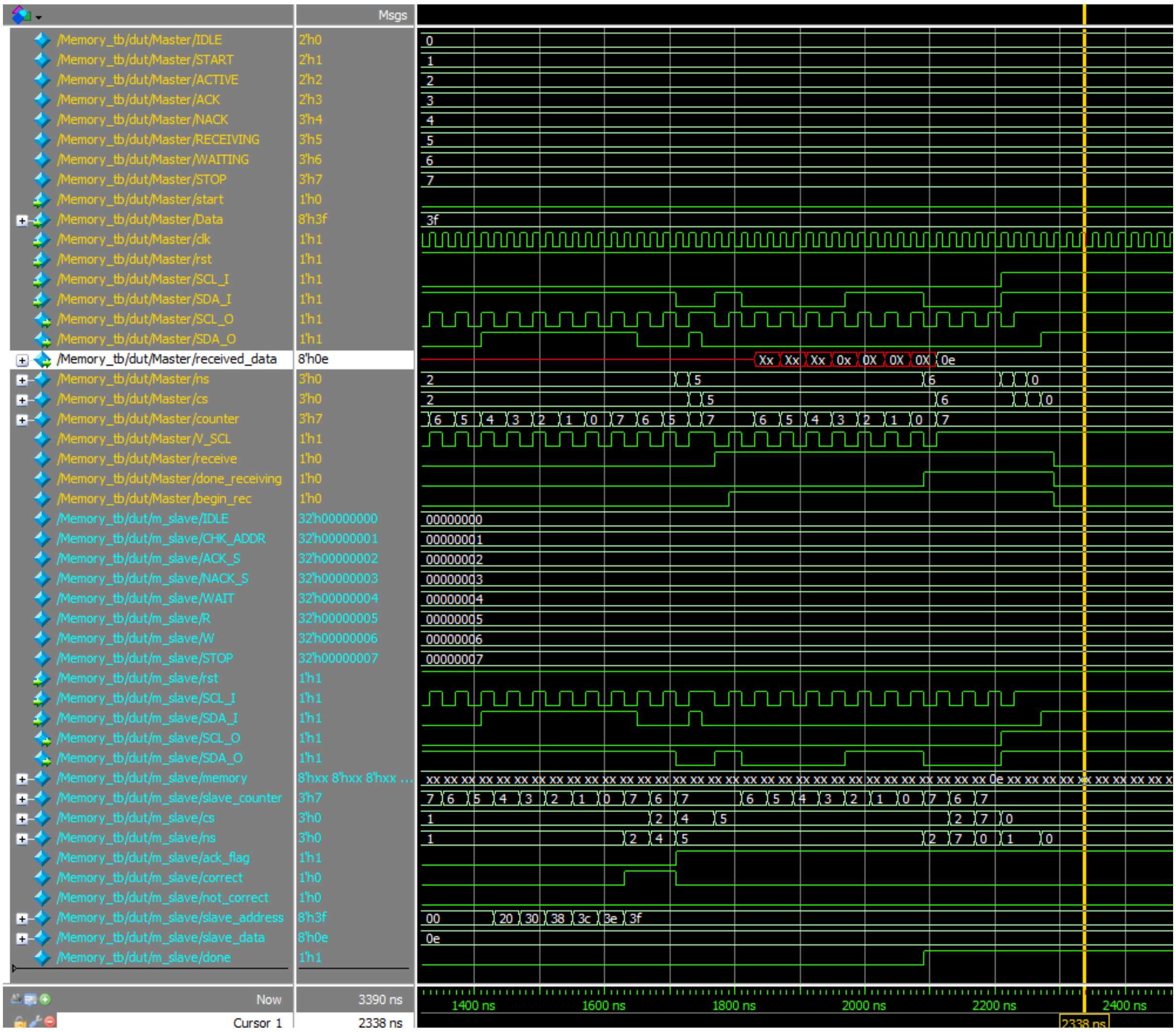
first scenario is to write data into the memory of our memory slave, as we see the slave address is the same as Data (7'b0011111) so we received the address correctly and also we received the data to be written correctly in slave_data signal



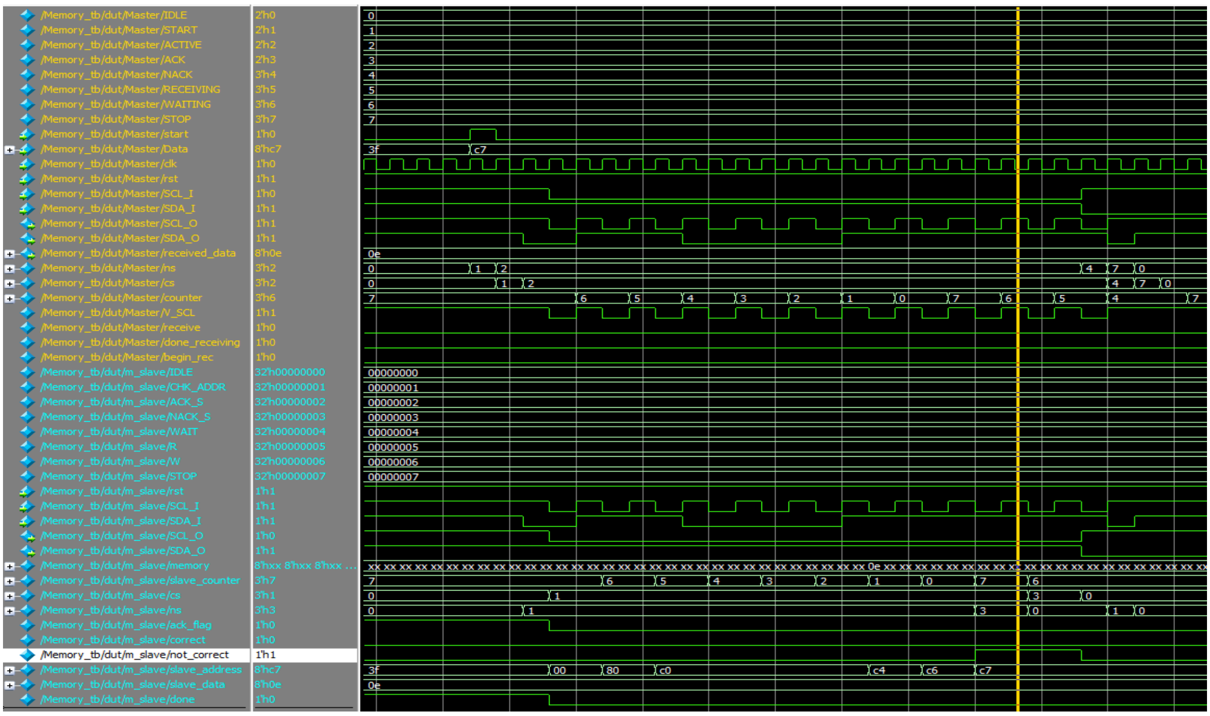
data in the memory is same as data in slave_data so the first scenario is done



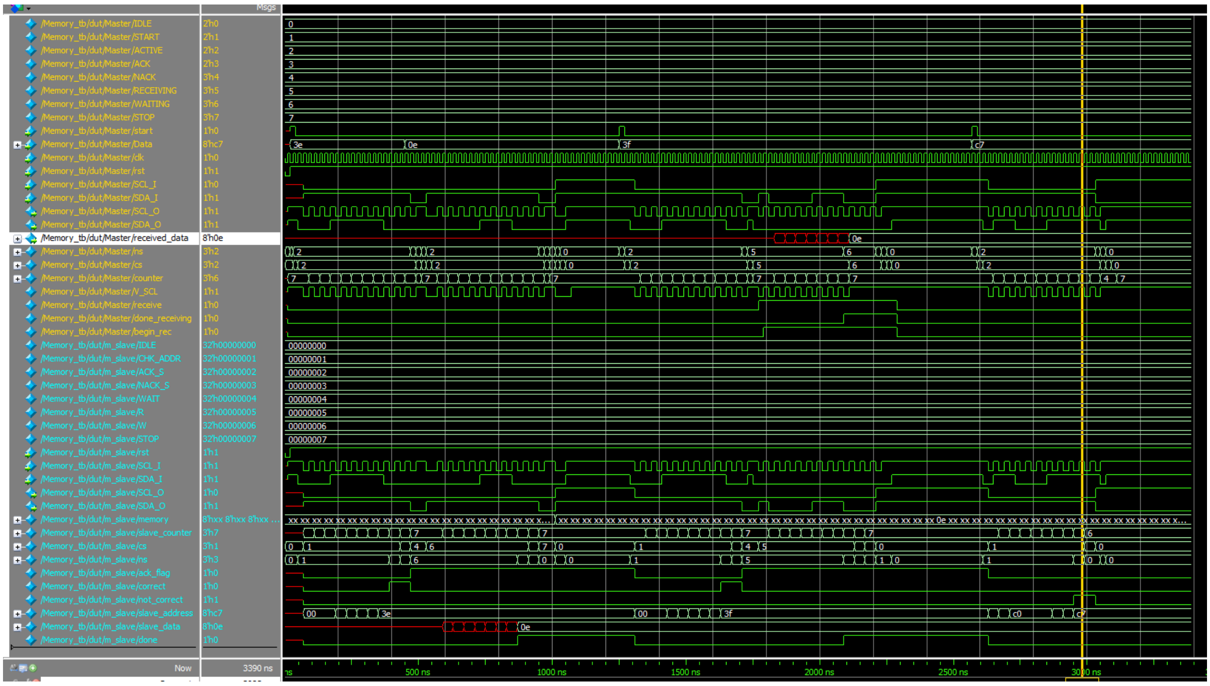
second scenario is to read data from the slave's memory with the same address we wrote data in, so received_data in our master has the same data we wrote into the memory



third scenario is to insert a wrong address to check whether slave responses or not, and as we see once slave got the wrong address, not_correct signal is high and slave goes to IDLE state



Full wave of **Memory_tb.sv** (test bench for memory slave only)

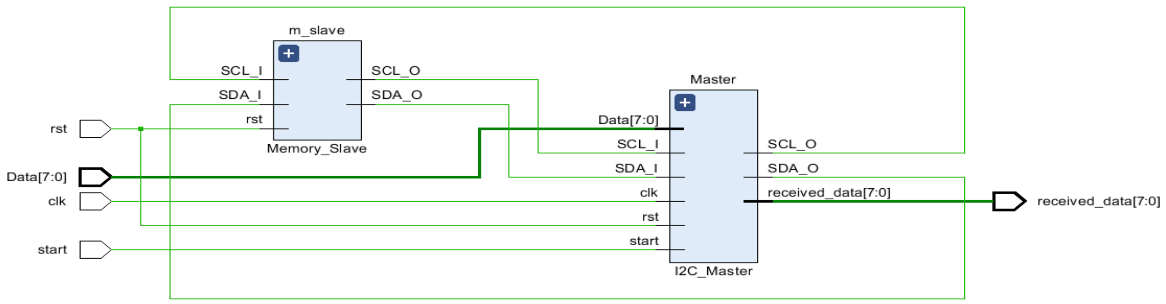


I2C_Wrapper.v > I2C_Wrapper

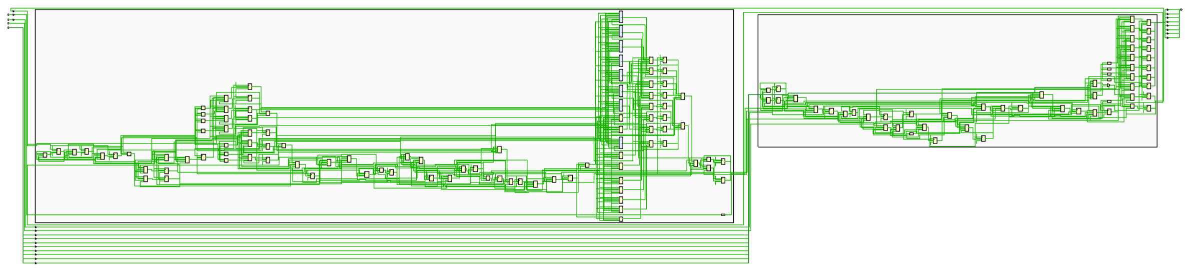
```verilog
// SCL & SDA signals are not special and they are used for every slave, you should uncomment
module I2C_Wrapper (
    input start,
    input [7:0] Data,
    input clk, rst,
    output [7:0] received_data // data comes to master from slave when it's a read operation
);
    // Signals between Master and Slaves
    // Master
    wire SDA_M_S; // Master to slave
    wire SDA_S_M; // Slave to master
    wire SCL_M_S; // From master to slaves
    wire SCL_S_M; // From slaves to master

    // Master Instantiation
    I2C_Master Master(
        .start(start),
        .Data(Data),
        .clk(clk),
        .rst(rst),
        .SCL_I(SCL_S_M),
        .SDA_I(SDA_S_M),
        .SCL_O(SCL_M_S),
        .SDA_O(SDA_M_S),
        .received_data(received_data)
    );

    // Memory Instantiation
    Memory_Slave m_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );

    /* // FIFO Instantiation
    FIFO_Slave f_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );

    / LIFO Instantiation
    LIFO_Slave l_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );*/
endmodule
```

Note: if you want to check on Memory_Slave, you should comment the other two slaves just like in the snapshot, this acts like if we make the other two slaves in a high-impedance state so they don't interrupt the demanded slave (Memory_Slave)

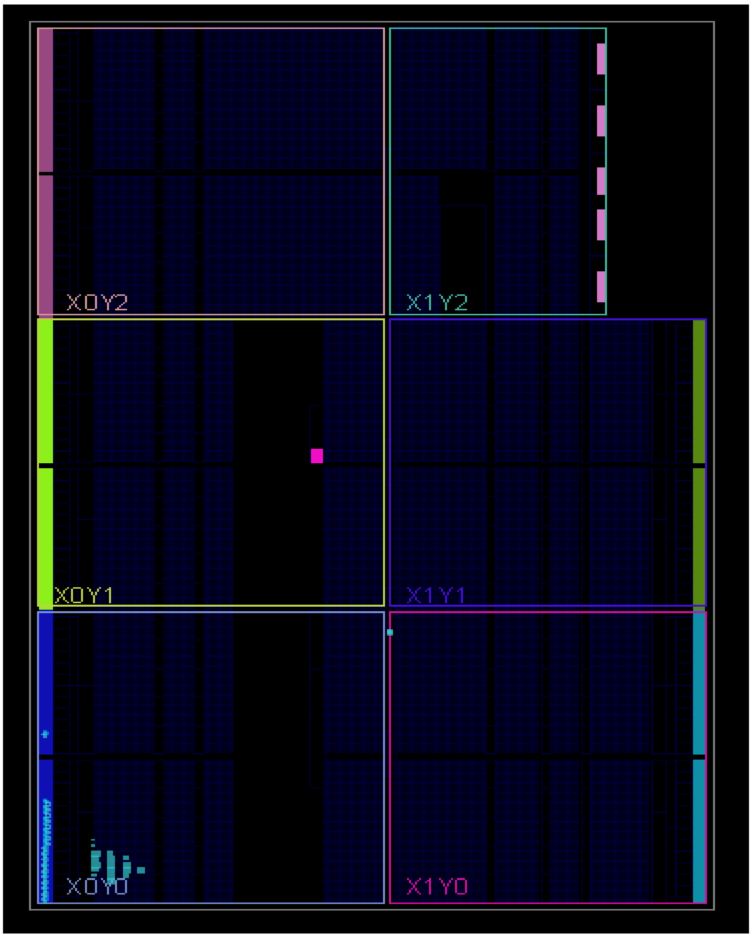
- **Vivado:**
  - Elaboration



  - Synthesis



  - Implementation

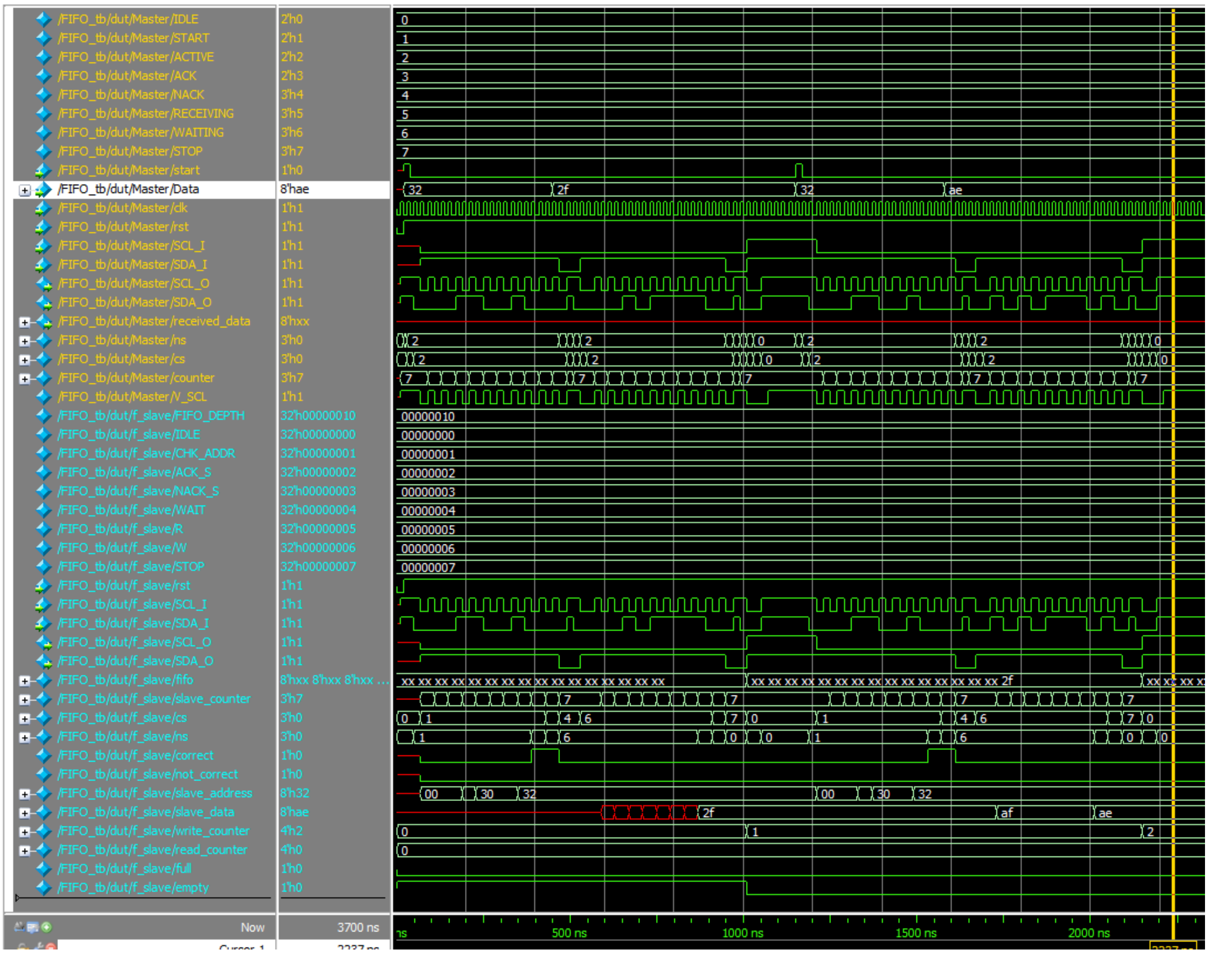


Messages showing no critical messages after implementation

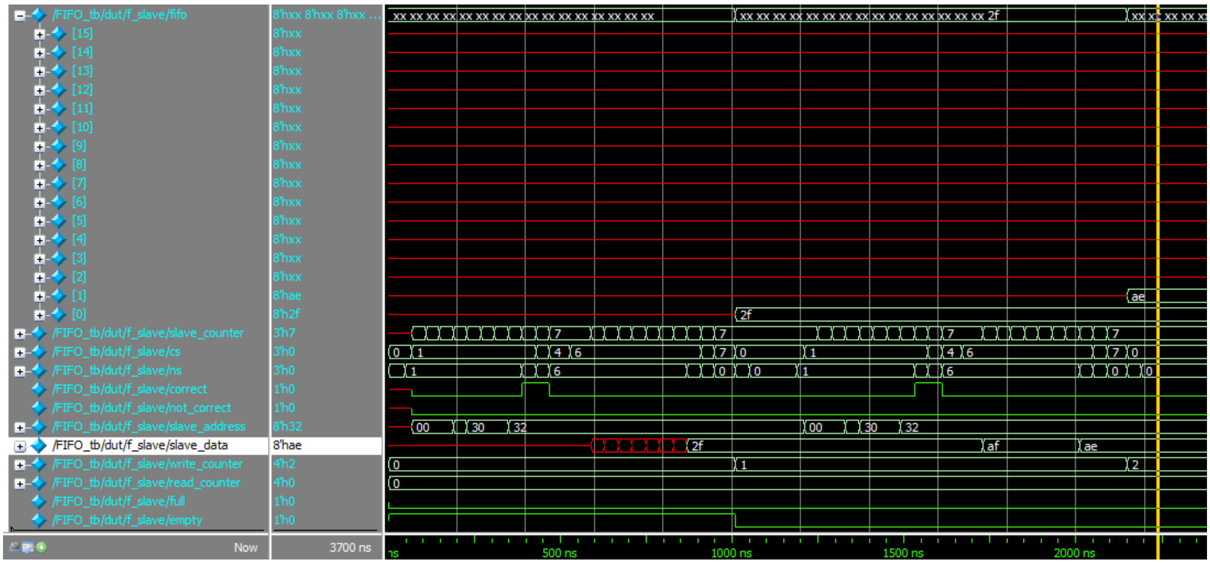
FIFO

- ## Verifying Functionality:

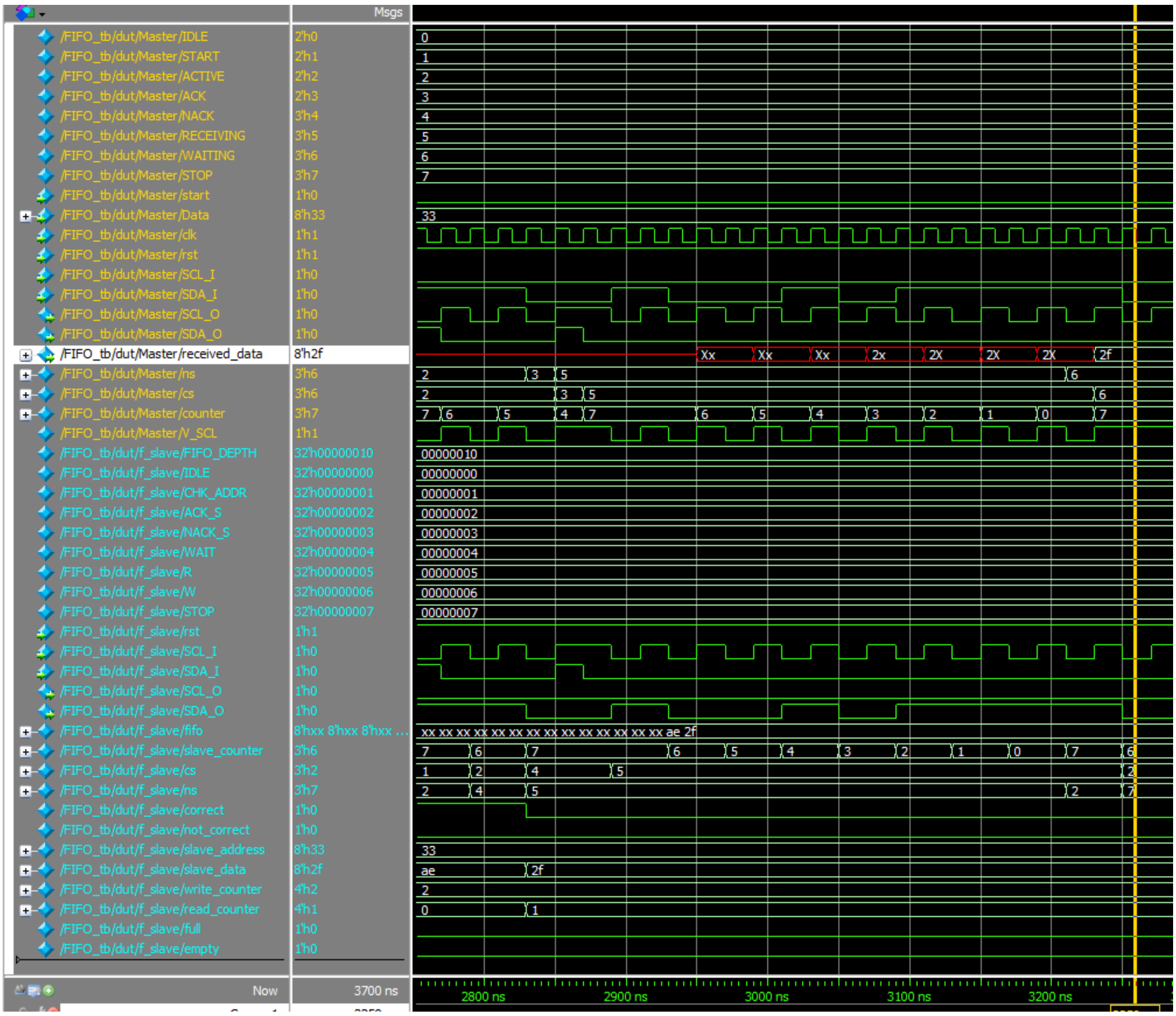
fourth scenario is to write two values into our FIFO slave with its unique address (7'b0011001), so as we see the slave_data signal got the two values and they were written into the FIFO so write_counter is now 2



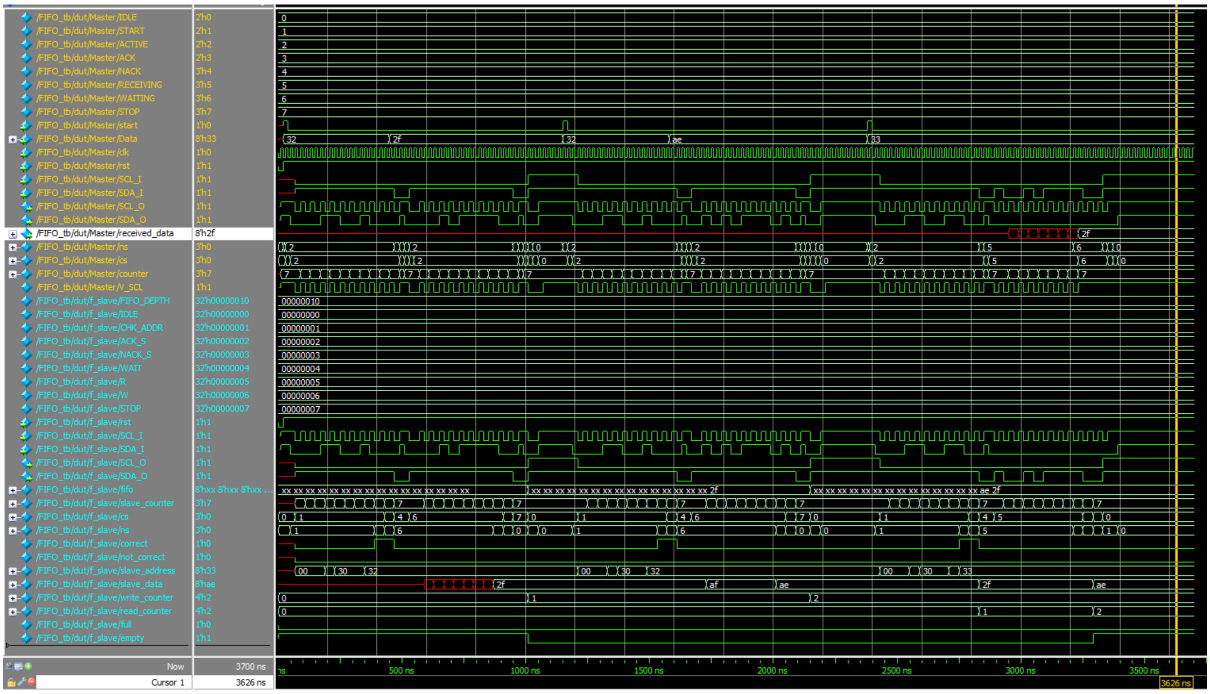
FIFO now has the two values in order



fifth scenario is to read data from the slave's FIFO, so we received the first value in the FIFO (First In First Out)



Full wave of **FIFO_tb.sv** (test bench for FIFO slave only)

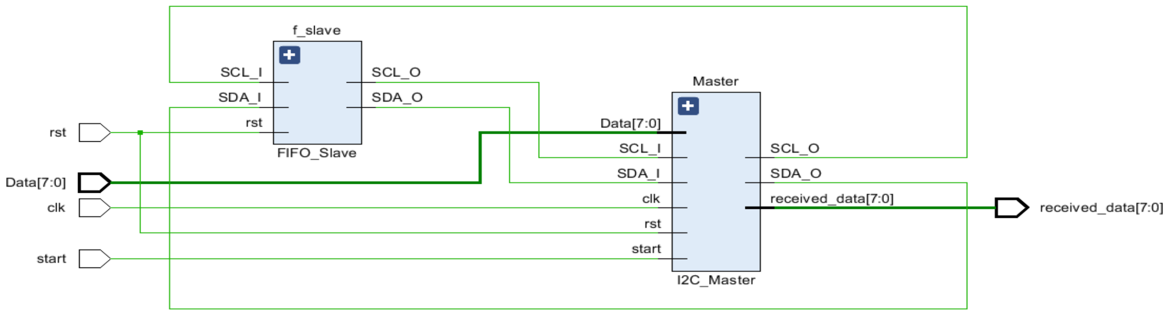


I2C_Wrapper.v > I2C_Wrapper

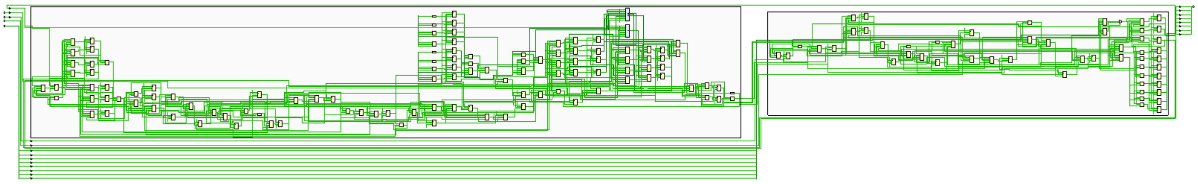
```verilog
// SCL & SDA signals are not special and they are used for every slave, you should uncomment t
module I2C_Wrapper (
    input start,
    input [7:0] Data,
    input clk, rst,
    output [7:0] received_data // data comes to master from slave when it's a read operation
);
    // Signals between Master and Slaves
    // Master
    wire SDA_M_S; // Master to slave
    wire SDA_S_M; // Slave to master
    wire SCL_M_S; // From master to slaves
    wire SCL_S_M; // From slaves to master

    // Master Instantiation
    I2C_Master Master(
        .start(start),
        .Data(Data),
        .clk(clk),
        .rst(rst),
        .SCL_I(SCL_S_M),
        .SDA_I(SDA_S_M),
        .SCL_O(SCL_M_S),
        .SDA_O(SDA_M_S),
        .received_data(received_data)
    );

    /*// Memory Instantiation
    Memory_Slave m_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );*/

    // FIFO Instantiation
    FIFO_Slave f_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );

    /* LIFO Instantiation
    LIFO_Slave l_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );*/
endmodule
```

Note: if you want to check on FIFO_Slave, you should comment the other two slaves just like in the snapshot, this acts like if we make the other two slaves in a high-impedance state so they don't interrupt the demanded slave (FIFO_Slave)

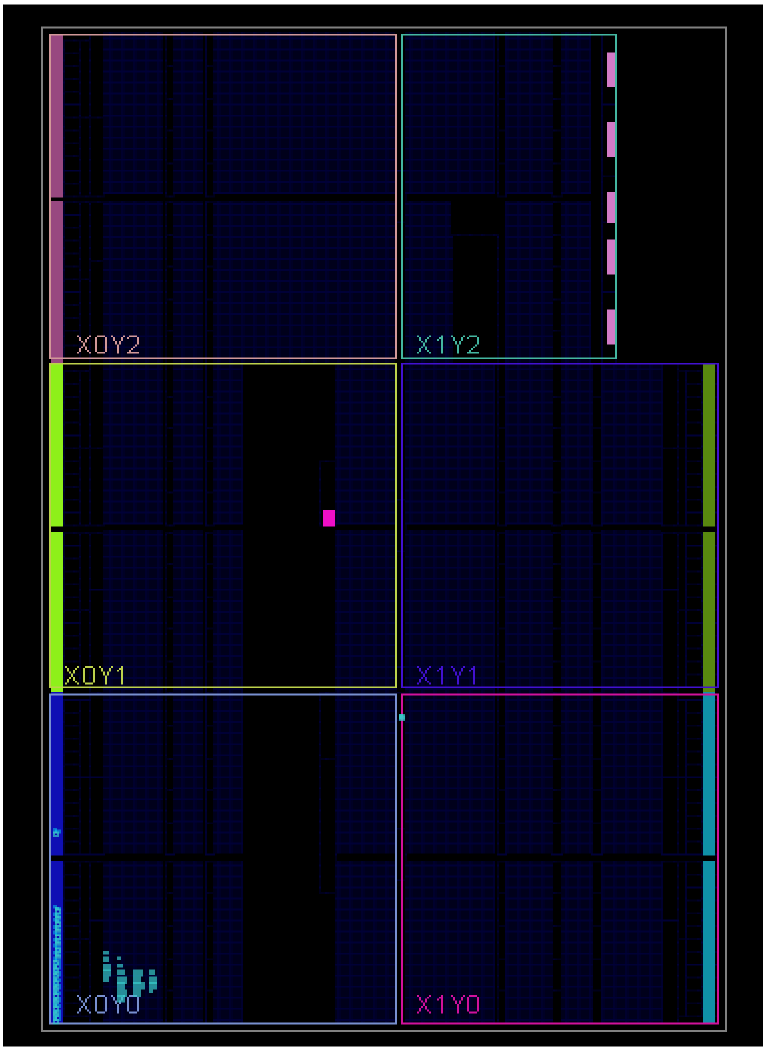
- **Vivado:**
  - Elaboration



  - Synthesis



  - Implementation

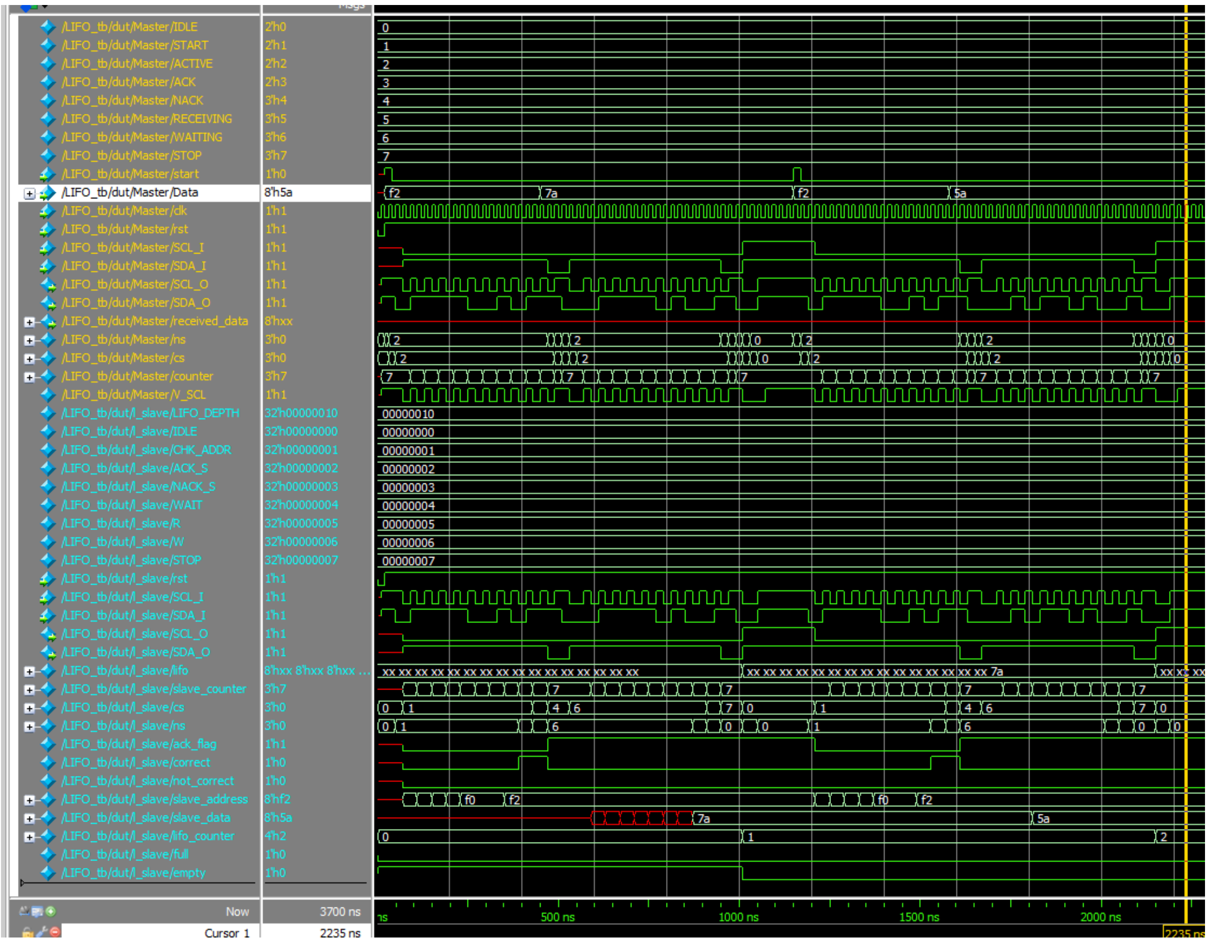


Messages showing no critical messages after implementation

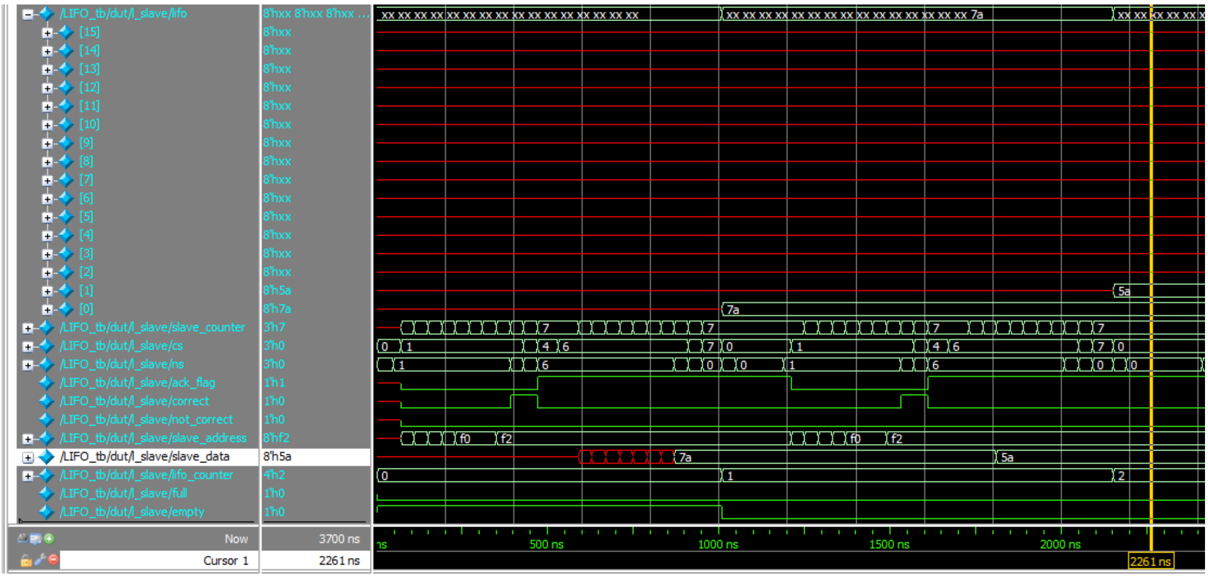
LIFO

- ## Verifying Functionality:

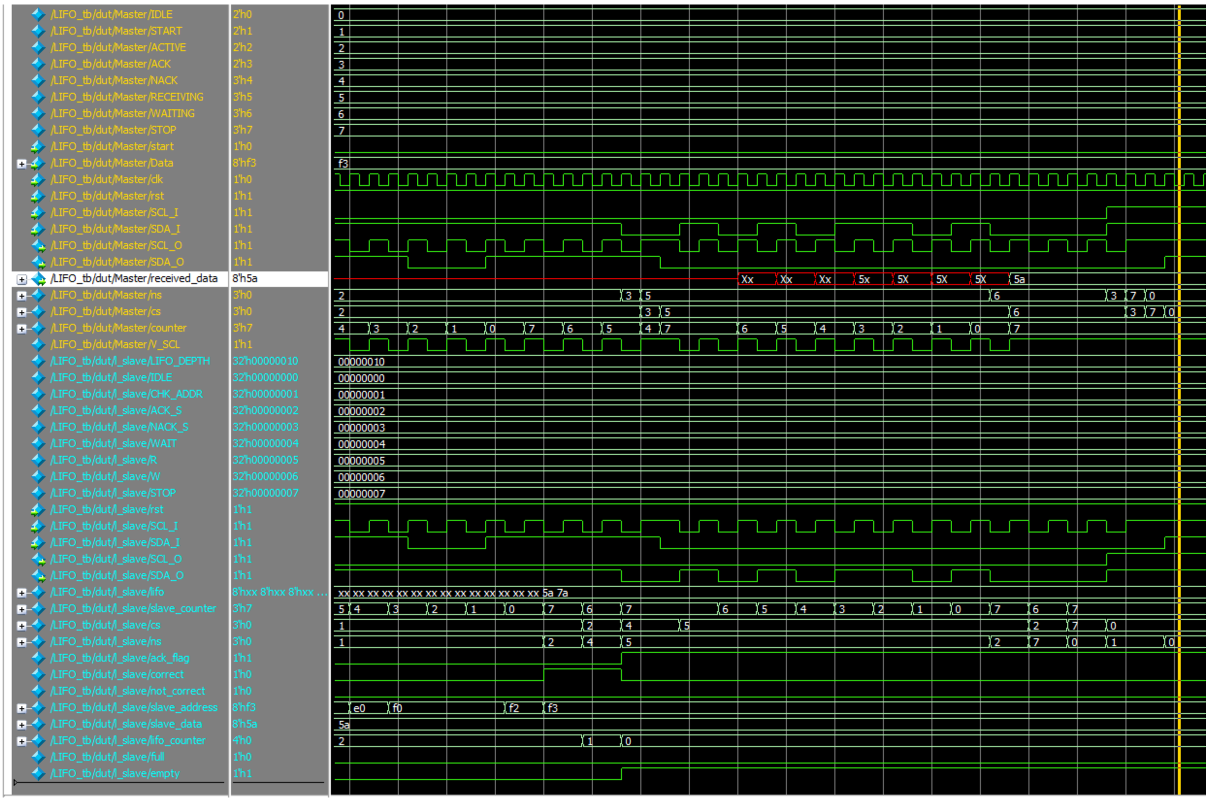
sixth scenario is to write two values into our LIFO slave with it's unique address (7'b1111001), so as we see the slave_data signal got the two values and they were written into the LIFO so write_counter is now 2



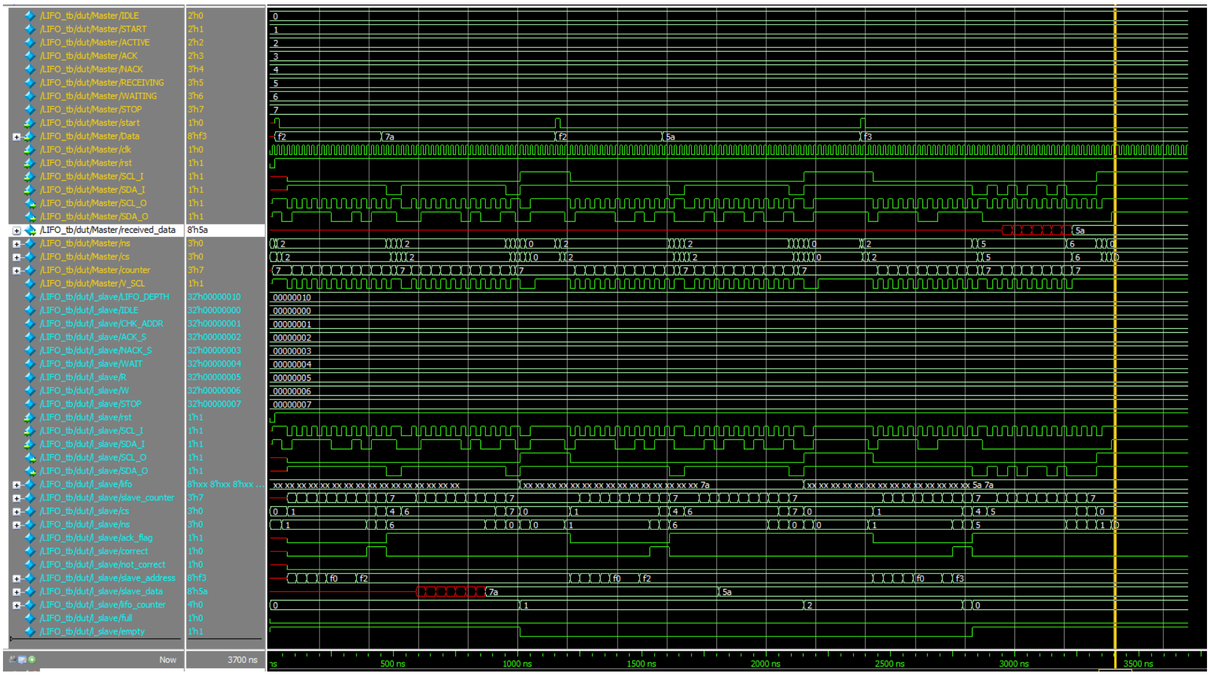
LIFO now has the two values in order



seventh scenario is to read data from the slave's LIFO, so we received the last value in the LIFO (Last In First Out)



Full wave of **LIFO_tb.sv** (test bench for LIFO slave only)

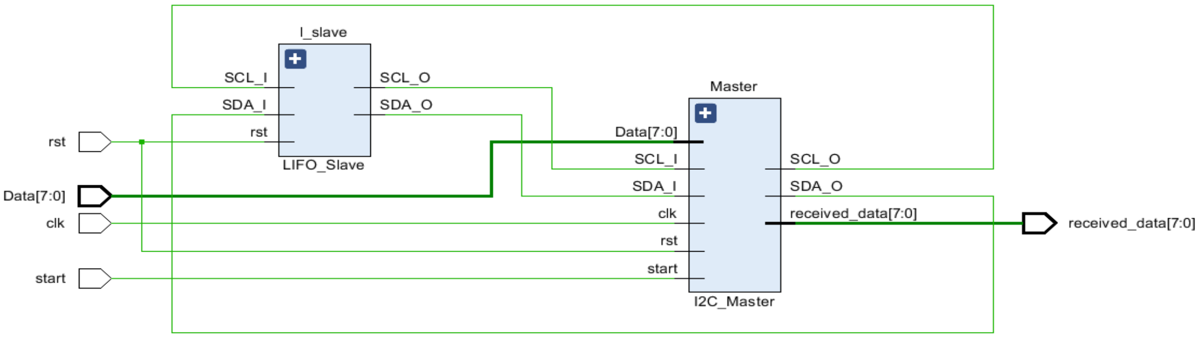


I2C_Wrapper.v > I2C_Wrapper

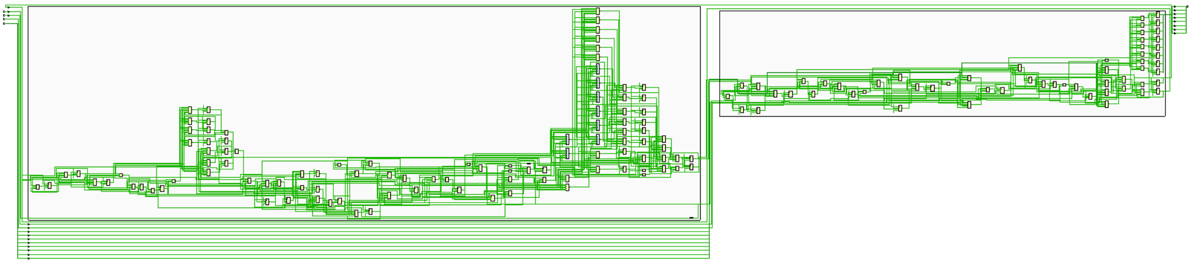
```verilog
// SCL & SDA signals are not special and they are used for every slave, you should uncomment t
module I2C_Wrapper (
    input start,
    input [7:0] Data,
    input clk, rst,
    output [7:0] received_data // data comes to master from slave when it's a read operation
);
    // Signals between Master and Slaves
    // Master
    wire SDA_M_S; // Master to slave
    wire SDA_S_M; // Slave to master
    wire SCL_M_S; // From master to slaves
    wire SCL_S_M; // From slaves to master

    // Master Instantiation
    I2C_Master Master(
        .start(start),
        .Data(Data),
        .clk(clk),
        .rst(rst),
        .SCL_I(SCL_S_M),
        .SDA_I(SDA_S_M),
        .SCL_O(SCL_M_S),
        .SDA_O(SDA_M_S),
        .received_data(received_data)
    );

    /*// Memory Instantiation
    Memory_Slave m_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );

    // FIFO Instantiation
    FIFO_Slave f_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );*/

    // LIFO Instantiation
    LIFO_Slave l_slave(
        .rst(rst),
        .SCL_I(SCL_M_S),
        .SDA_I(SDA_M_S),
        .SCL_O(SCL_S_M),
        .SDA_O(SDA_S_M)
    );
endmodule
```

Note: if you want to check on LIFO_Slave, you should comment the other two slaves just like in the snapshot, this acts like if we make the other two slaves in a high-impedance state so they don't interrupt the demanded slave (LIFO_Slave)

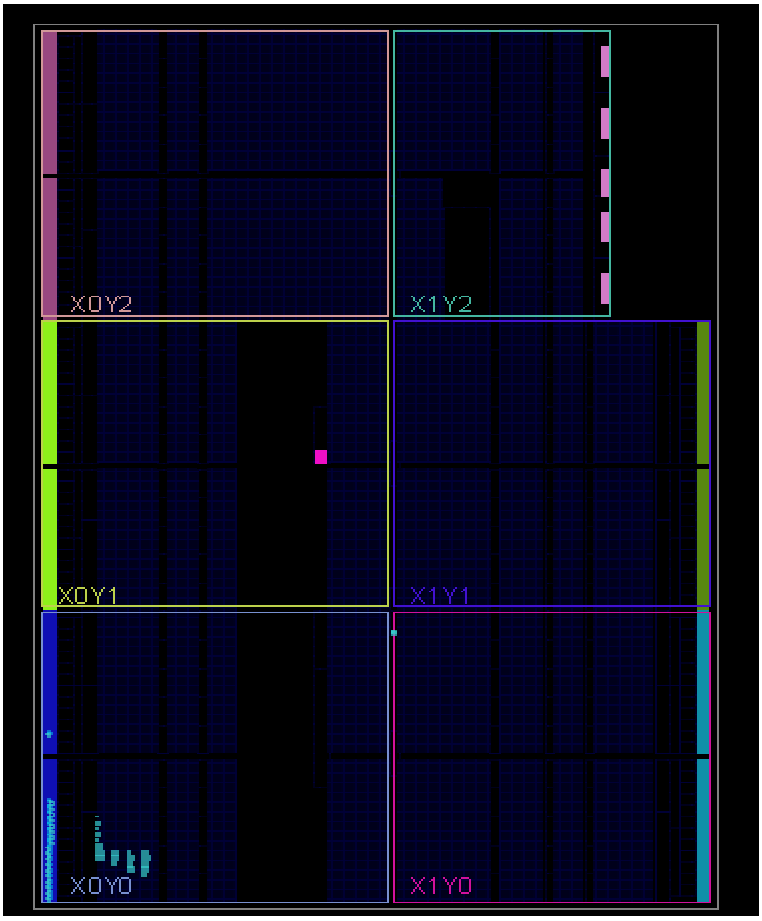
- **Vivado:**
  - Elaboration



  - Synthesis



  - Implementation



Messages showing no critical messages after implementation

Conclusion

The I2C bus implementation with three distinct slave devices—Memory, FIFO, and LIFO—will be thoroughly verified to ensure accurate and reliable communication. Each slave will undergo **individual functional verification**, followed by **elaboration, synthesis, and implementation** using **Vivado**, ensuring seamless integration into the hardware design.

Simulation snapshots generated in **QuestaSim** will distinctly identify the **Master** in **gold** and the **Slaves** in **blue**, providing a clear visual representation of the interaction between the components. This comprehensive process guarantees that each slave operates correctly within the I2C protocol framework, paving the way for a robust, efficient design.